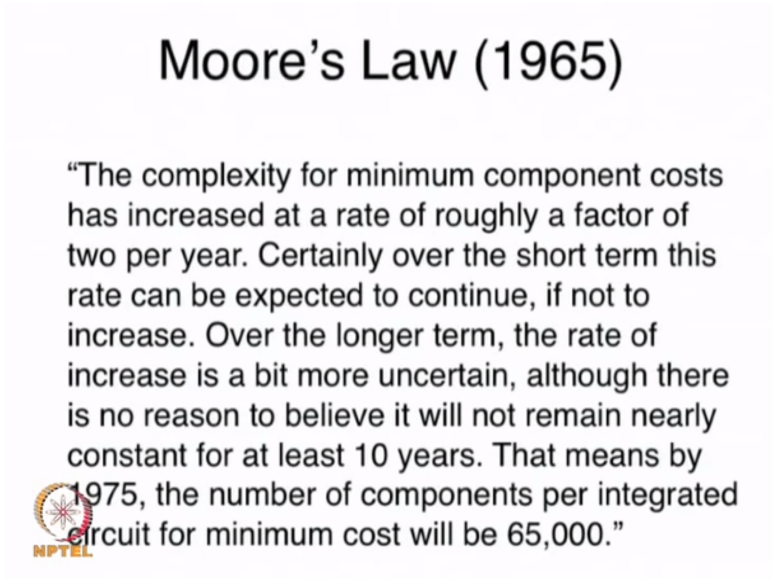
Parallel Computing

Prof. Subodh Kumar
Department of Computer Science & Engineering
Indian Institute of Technology – Delhi

Module No # 01
Lecture No # 01
Introduction

You have seen the contents of the course on the website very quickly you will see the mix of practical aspects of parallel programming and development of theory of parallel algorithms models of parallel computation. So you will get some theoretical back ground as well as learnt to practice on parallel programming on modern parallel computers.

(Refer Slide Time: 01:00)

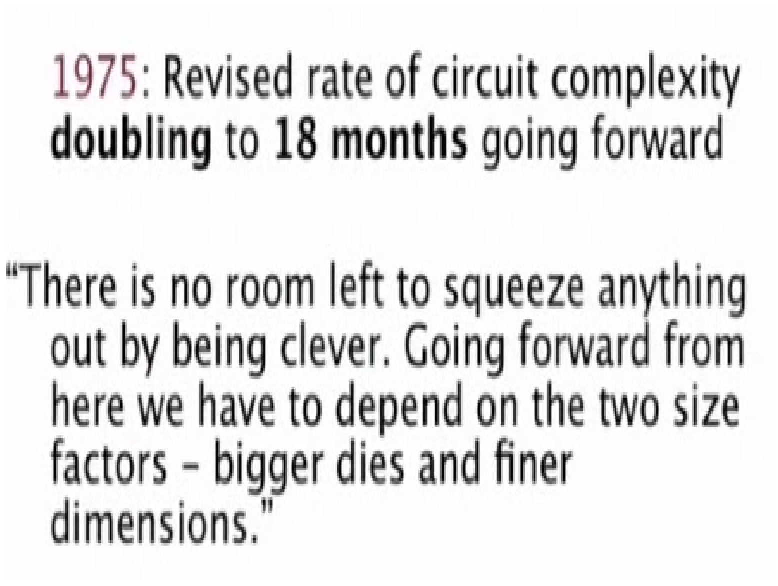


Moore's Law (1965)

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000."

Let us being with some motivation this is what is famously known as the Moore's law a statement by MOORE in 1965 which predicted over the next 10 years computers will become very fast when those 10 years elapsed.

(Refer Slide Time: 01:19)



1975: Revised rate of circuit complexity **doubling** to **18 months** going forward

"There is no room left to squeeze anything out by being clever. Going forward from here we have to depend on the two size factors – bigger dies and finer dimensions."

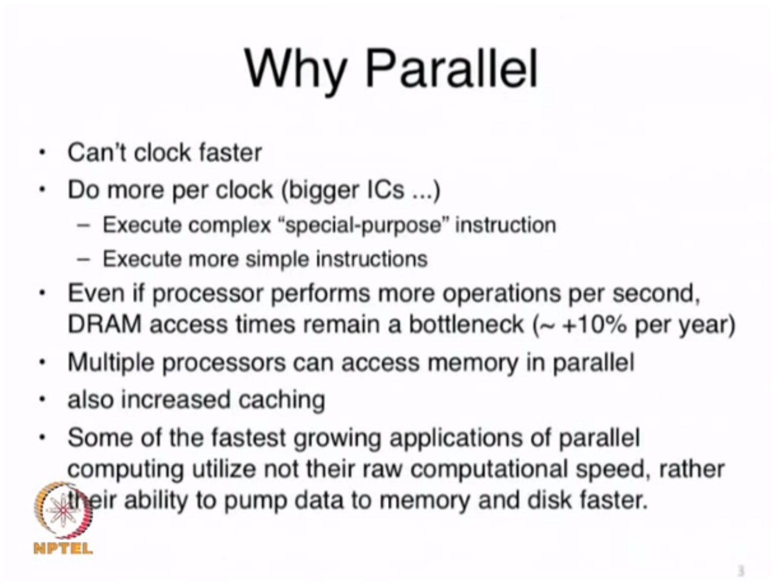
It turned out that computers continued to become very fast and it is basic been revised to saying that the computer speed is going to be essentially doubling every 18 months. And that happen by variety of techniques in the beginning that estimate of 24 months 12 months to which was the 1965 statement to double the speed was based on the fact that there is lot of innovation in science and engineering to be done to make computers go faster.

In 1975 Moore predicted that we have essentially saturated out intelligence we cannot make innovation into making things better and faster but still just by advent of technology of being able to lay thinner wires on silicon we will be able to pack in more and more and more transistors which will in turn make computers faster ok.

And so there are two parts to it one was that you can drive the clocks much faster drive the gates using a faster clock which as you probably have seen as saturated in last few years you do not see the nineties style going from 1 gigahertz to 1.5 gigahertz to 2 gigahertz computers on. But still new computers keep coming on and its is because the law still holds even today you are able to does not really say about fast it says about you can pack more circuitry in the same amount of space okay.

And that as continued to hold the challenge is make that circuitry do things that you want to do okay and part of the speed up happened because of increasing clock cycle speed but part of it is happening because we have much more compute engines on the chip that can do many things at the same time okay.

(Refer Slide Time: 03:35)



Why Parallel

- Can't clock faster
- Do more per clock (bigger ICs ...)
  - Execute complex "special-purpose" instruction
  - Execute more simple instructions
- Even if processor performs more operations per second, DRAM access times remain a bottleneck (~ +10% per year)
- Multiple processors can access memory in parallel
- also increased caching
- Some of the fastest growing applications of parallel computing utilize not their raw computational speed, rather their ability to pump data to memory and disk faster.

So that is the one reason that parallel that kind of started to become popular there used to be a niche area for a long time there even used to be we will go through the history a little bit they used to be massively parallel computers which practically less compute less of a compute engine than PC's of today.

Being sold for a millions of dollars at the time but that kind of work died down around nineties and then which the advent of INTEL's PC based or rather Intel based chips for microprocessors let into a slightly different directions than the earlier direction. And now Intel itself trying to pack more and more course into a single chip.

So that you can do more at the same time but only being able to pack more course does not always help you have to be able to use those course to actually do things at the same time that together mean something okay. Having clocks running at 24 course or 100 course or 1000 course does not give you any computation at the end of the day each clock we have to do some work that can whose result can be used later on by somebody else right.

So that essentially what we need to study so in short clocks stopped getting faster but you can in fact pack more and more information more and more circuitry more and more data parallel paths on single chips. Chips are getting bigger in terms of how much how many gates they are not in terms necessary although they are getting bigger in size also somewhat especially NVidia keeps increasing the size of its chips.

But it is more of the packing density on the chips that is going up which is causing many more compute engines to be available at the same time which we hopefully would be able to use in a useful way if you are able to do parallel programming efficiently. And even if we have all this processors you can make the processors go faster and faster and say this processors is 10 core processor is 20 core processor which can somehow internally figure out how to parallelize your application.

So when you have programming that is list of instructions it takes the next 100 instructions and execute in parallel like there have been architectures where you essentially compute without too much analysis of what instruction need to happen before what other instructions which is predictively run many instructions and some of those instructions will not have their data because they depend on previous instructions that have not happened yet and you throw away the results of the instructions.

You just use the results of instructions that had their data at the time they were on so you that is in some sense brute force parallelism can take you only so far. But even if you were able to make the processor faster and faster that is not going to be the end of the story. Because ultimately you need data on which the processers will be doing something you need operands on which the operations will happen okay.

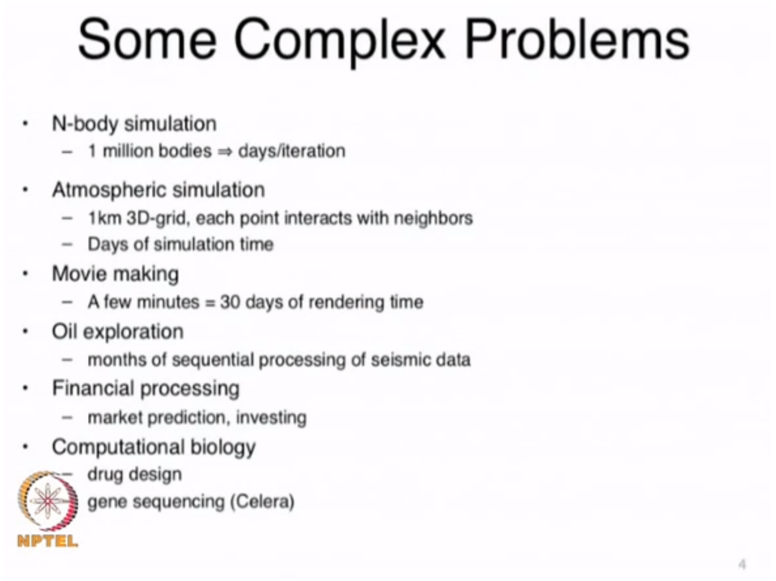
And these operands will stay in the memory DRAM or probably come from network through some IO interface and those are not going up in speed nearly as fast as processors while processor speed although in again using it in layman's terms is going up by doubling every 18 months that DRAM speed is basically going up by 10% every here or so.

So the lag is only going higher and higher and higher all right making good use of parallel architectures and parallel programs also helps you with that disconnect because now because single DRAM chip cannot be access faster enough with a processor that is running really fast. But if you have 100 of processors connected to hundreds of DRAM chips each can probably mashed with the speed of the DRAM okay.

Because each processor is slightly slower than the overall the number of operations per second that you can run all right. So that is another reason why parallel programming has started to grow because you sip the architecture simply cannot have a single CPU connected to single memory bus with s single memory hanging on the other side okay it just does not scale the memory becomes the bottle neck too quickly.

And in fact the Google's and yahoo's of the world are focusing on that side of the story how do you pump operands into the CPU fast enough so that the operation can be done and so some of the research has shifted into that interface between the CPU and rest of the architecture okay.

(Refer Slide Time: 09:27)



Some Complex Problems

- N-body simulation
  - 1 million bodies ⇒ days/iteration
- Atmospheric simulation
  - 1km 3D-grid, each point interacts with neighbors
  - Days of simulation time
- Movie making
  - A few minutes = 30 days of rendering time
- Oil exploration
  - months of sequential processing of seismic data
- Financial processing
  - market prediction, investing
- Computational biology
  - drug design
  - gene sequencing (Celera)

Let me also very quickly go through some sample applications you might consider using these as kind of test beds or target applications for you projects at the end of the course. These are some of the areas not all some of the areas where high speed parallel computation has become very popular because of the need for it because the amount of computation that is needed is extremely high and old they were always being done on super computers and super computers.

Where in some sense on the leading edge of being to parallel program but now that technology has become available to everybody that kind of skill is required in a much more wider context okay. And so here are some of examples of fast these application would run this is probably 2 to 3 years old data at this time and body simulation with a million bodies and again it is depends on the simulation.

So do not take the exact number without any grain of salt because there is lot of information that is not being provided to you it is just at a high level is one example of N body simulation that runs at days per iteration okay. Atmospheric simulation whether simulation is say extremely let us say hot topic recently because of global warming simulation of how atmosphere will be behave 50 years from now 100 years from now is what people are doing on computers okay.

And being so you have probably seen how well we can predict tomorrow's temperature so that is because our models are still evolving and we have to predict tomorrow's temperature today we cannot wait until day after tomorrow for the model to complete it is computation and then say yesterday temperature was 7 degrees okay. And so there is lot of computation being thrown there but still there is lot more to be done.

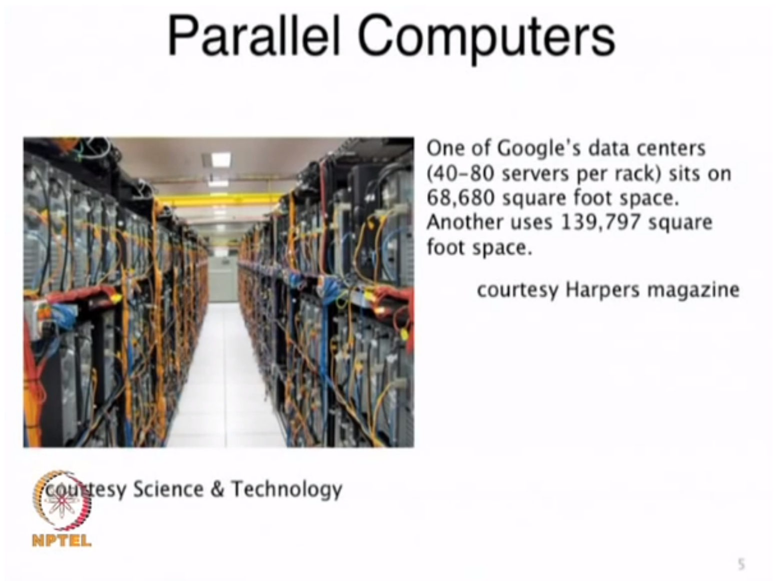
The entire area of weather simulation atmospheric simulation has seen massive application of large number of super computers or multi-core machines people are using it on moviemaking all the time computer graphics at a very high quality does not come cheap and so and that happens to be a problem that is easily parallelizable right.

The generation of a movie essentially generation of many images in a sequence and you can generate an image of 2 minutes later in the movie at the same time as you are generating the image of now. And so that is what is sometimes called embarrassing parallel because there is you just as many computers as you have you can use them all without worrying about what come before what comes after whose result needs to go to whom all that information or headache is not there in that problem.

But still movie making on cluster of machine still taking somewhere on few minutes depending on how complex the scene is to a month of a time per minute for highly complex scenes of animation movies okay. Oil exploration is another area where because the payoff is so high people want to get their first people want to make the computations about fitting of model in terms of where that oil may be found in the vast expanses of ocean or earth is another place where people seeing lot of explosion of parallel programming.

Stock pricing again high payoff potentially lot of data centers on the wall street and the lot of people here using those data center again computation biology bio informatics in general has a lot of simulation of genes sequencing of genes how genes would evolve and things of that sort again lot of computation is needed and naturally people are moving towards using at least moderate sized if not massive size data centers for those cases.

(Refer Slide Time: 14:18)



Parallel Computers

One of Google's data centers (40-80 servers per rack) sits on 68,680 square foot space. Another uses 139,797 square foot space.

courtesy Harpers magazine

courtesy Science & Technology

Here is one picture of one row of rather two row of racks in a data center of unknown named company this is not Google data center which this information is coming from somewhere else. Google's data center one of Google's data center is 68,000 square foot another one is 140,000 square foot and each of these things going vertically is a single rack and then there is sequence of racks that is in this photograph and there is probably something close to 40 or 50 racks in a row.

Each rack depending on the age or when it was built is going to have 40 to somewhere close to 80 CPU boxes. CPU slash GPU boxes some server and so just imagine so much of space filled volumetric area and height with processing that needs to be controlled needs to do something useful of course typical usage of these data centers are in that context of embarrassingly parallel.

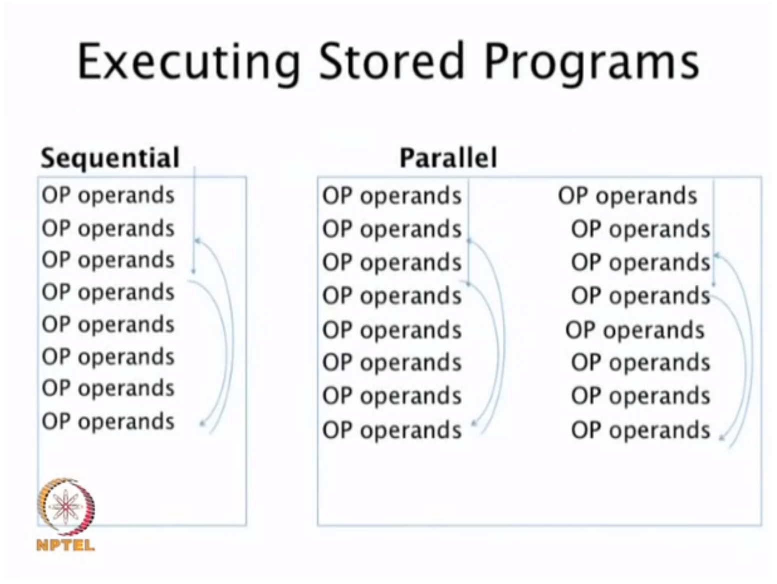
Because there are lots of users trying to do different things and so there are technologies like virtualization which takes this data center breaks it up into logical machines that each user is basically using on his own or own and so they are really being massively parallelized and was single application is not run on the entire data center typical computation as well as storage typically.

For example a Google data center will have with each note enough number of CPU board and enough number of disks it is a data center these are and in fact it is it is less of an optimization of space as much of the interconnect and the power which is where Google makes it own no the fact is that they are still buying these 1U boxes from the market and putting it there right. So that 1U boxes is going to take its space you can do nothing about that.

In this case it is not 1U boxes but typical data or an example of a data center we just buying a 1U boxes and putting them up in a rack all right. And in fact that number is coming from 40 to 80 servers is because there are these two sided racks which can hold 20 1U boxes on one side and 20 1U boxes on other side. Recently TNA1 which was listed as the fastest super computer in the world which is a made of 14,000 Intel Xeon processors.

And some 7000 NVidia Tesla GPU working together in this case it is not like a data center where there are lots of independent processes is independent programs running. Single program is being speeded by using all of these different CPU’s and GPU’s at the same time which is more in line with what we are going to be talking about in this course all right.

(Refer Slide Time: 17:56)


Executing Stored Programs

Let us move on to thinking about so this is just a very brief update so to speak on those of you who have done and even those of you have not done OS and into the computer type stuff. We execute stored programs on computers of today typically and in a sequential context you have got a list of instructions one after the other and there is an CPU execution engine which is going to read the operation of code and the operands and perform that okay.

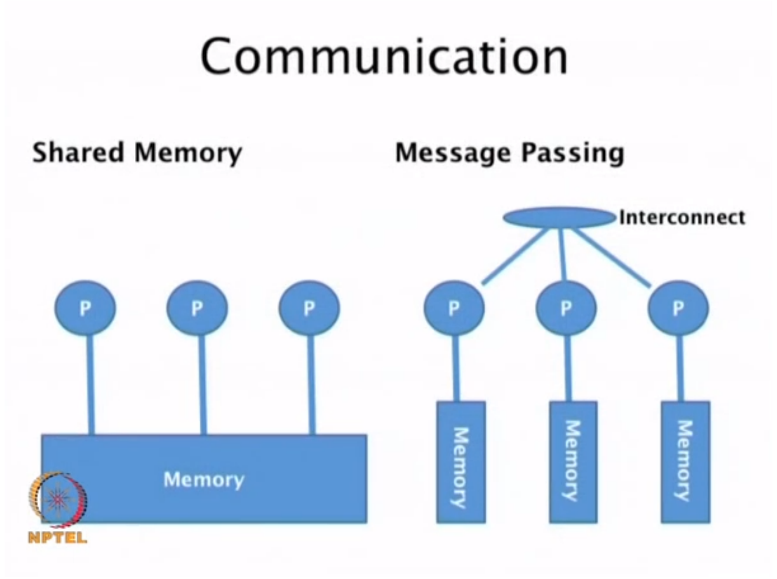
And there is some fetch and execute cycle but typically it is going to go one after the other type one instruction execute then the next is execute in the next executed and of course there is some notion of jumping and looping and all that right you can go forward you can go backward but typically one instruction at a time whereas in the parallel context you have independent streams in this instructions and that is really the basic difference.

Independent steams being executed independently at the same time two different PC's program counters being incremented at the same time there is no saying where this program counter is with respect to that program counter line this might have been implemented 5 times and that might be implemented 10 times. We are not going to make any assumptions about how synchronized they are or unsynchronized they are okay.

So each one is going to do whatever it does in a sequential context but it happens at the same time strictly speaking this is although I am saying that is parallel this is sequential even its equation machines we have this notion of non-determinism because even though at the processor level there is a sequence of instructions and it executes one after the other at the application level there are many different processes many different threads and context switches among them okay.

And if these threads or processes have any knowledge of each other if they communicate with each other then it boils down to the same thing because there are two independent threads you do not know how many instructions of this thread as done how many instructions that thread as done and you still needs to somehow be able to manage them figure out what data that this produces can be used consumed by that other thread okay.

(Refer Slide Time: 20:50)



So although slide of the right hand sides said parallel strictly it is concurrent okay all right and in terms of communication there are two styles of communications so to speak that are followed. In the older days they used to be you have a shared memory computer or you have a message passing computer these days you have a shared memory and a message passing computer okay.

Most practical machines on which you will do parallel programming will have some notion of shared memory as well as some notion of communicating with some other processors share memory also one way of communication. You put some data in shared memory the other processors reads it from there. So if communicate with that but shared memory means a shared address space so when you say I am writing this data to variable number variable named alpha and the other thread needs a value of alpha.

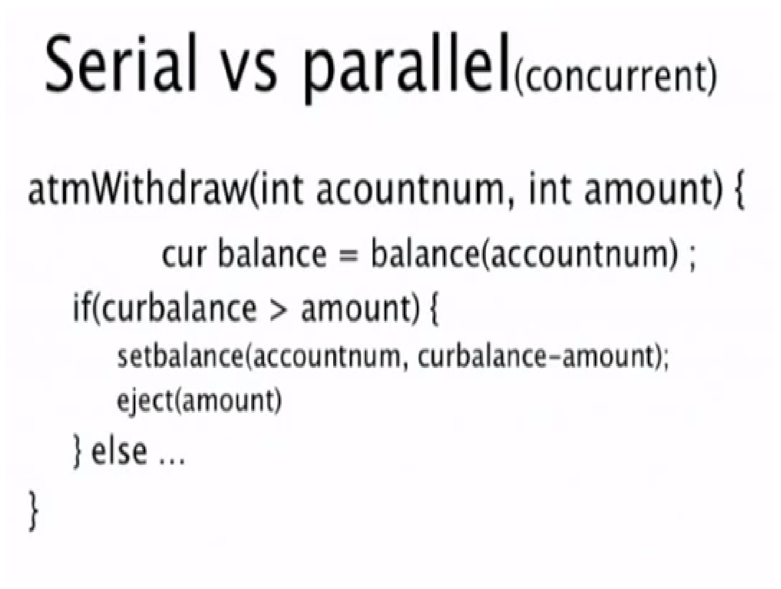
It can figure out the alpha go to that address and read it right so there has to be some notion of shared address space also in message passing you basically have some network I have got a pocket of data there is no address on it there is an address on the receiver and I say this data needs to go to that receiver and the receiver if the receiver is willing to accept that data from the sender then the handsome protocol needs to be followed some handshake needs to be followed.

And that data will get from the sender to the receiver and the communication has happened okay open MP is going to be basically on the left side my open MP is using the shared memory model only an MPI which is short for message passing interface is going to be on the right hand side okay. And you can use open MP as well as MPI in same program probably because 2 course two of the course that need to interact do not have the shared memory.

Or even if they have a shared memory for some reason you decide not to use the shared memory not to share the address space and you send the data by using some protocol and we are not going to get again as I had mentioned earlier into the types of protocols the types of issues with sending data just like to assume it we are going to say all of that is inbuilt in something like an MPI or open MPI toolkit which is going to assume it is there we are going to use it okay.

Sometimes we will care about how far something is because getting data to far away place takes more time but usually we will simply say this data needs to go one up to somebody else and so it takes one unit of time okay.

(Refer Slide Time: 23:58)



Serial vs parallel(concurrent)

```
atmWithdraw(int acountnum, int amount) {
        cur balance = balance(accountnum) ;
    if(curbalance > amount) {
        setbalance(accountnum, curbalance-amount);
        eject(amount)
    } else ...
}
```

Continuing on little more about the concurrent stuff so what happens if this program runs in a sequential context I want to be draw something from the ATM I say this is the amount of money to be draw say current balance is equal to query at database I get the balance of that account number and if the current balance is greater than that amount we set the balance to the old balance minus whatever we are be drawing and then eject that money from the ATM machine okay.

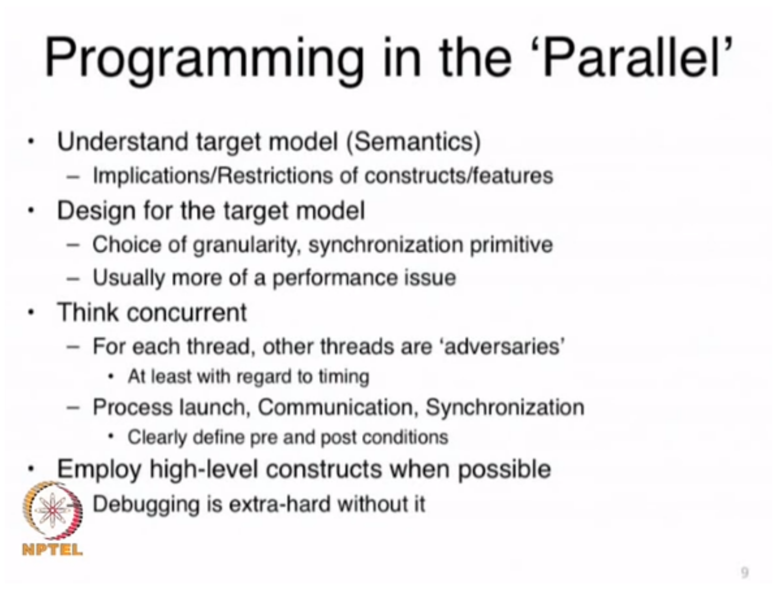
So this is proper program there is only on user one database but what needs to happen when you have many ATM's all connected to some notion of a logical database which may or may not be at the same place what will happen if this program runs race conditions right. So we have two ATM's right and you go and the query on the both ATM's for the same account number because you can get multiple ATM cards for the same account.

And you current balance is equal to two thousand rupees right and you say okay I can withdraw 500 both of them check current balance which is 1000 is greater than the amount I am drawing which is 500 so they set the balance to 1000 – 500 so the amount in the database says 500 and then you eject 500. So what happen you rejected 500 twice if the database got updated by the say it got updated twice in this case by the same value both of the said updated profile.

And so kinds of things that you need to moderate are what was shared here the current balance when you dead the query to the database we are querying about the same account number where is the notion of the shared and so whenever we have got multiple players trying to share a resource they are going to have to figure out how to manage the access to that resource like you have probably seen many ways of doing that in the OS course and those who have not done will see many ways of doing that in the west course.

But the concurrence is the number one concern when you doing parallel program figure out what depends on what is shared with what and then decide when one or the other can happen okay. If they can happen in parallel or concurrently then you allow them to happen if they are not going to give you correct result if they allow if they happen in any order then you force that order okay. So that is the number one concern while parallel programing and then other concerns we will get to shortly.

(Refer Slide Time: 27:19)



Programming in the 'Parallel'

- Understand target model (Semantics)
  - Implications/Restrictions of constructs/features
- Design for the target model
  - Choice of granularity, synchronization primitive
  - Usually more of a performance issue
- Think concurrent
  - For each thread, other threads are 'adversaries'
    - At least with regard to timing
  - Process launch, Communication, Synchronization
    - Clearly define pre and post conditions
- Employ high-level constructs when possible
  - Debugging is extra-hard without it

And so this is an expansion of that list of concern to be a good parallel programmer you are going to have to understand what the parallel architecture is right I said that we will assume a periodical model here I have got a set of processors and a memory that is shared across them and then I am going to give you an algorithm okay.

In order for that algorithm to work well on a given machine you are going to have to see how well it maps to that PRAM architecture and when it is different from PRAM architecture what are those differences and can those be hidden okay. So we have to be mindful of where the memory access are going to be slow where the data communication is going to take long and take that into account on top of the pyramid which does not account for those things.

And so you are always going to have the target model and we will design for the target model that is why I also said that when you do the assignment although your program will run fine probably correctly on your laptop but if there is a significant difference in the architect of the laptop versus the machine on which you eventually want to run it then even the algorithm might change.

But even the design choices that you made on the laptop may be different from what you have been made if you were running it on slightly different architecture okay. And again to stop repeating this until the end of the course think concurrent that is one common problem with people learning parallel programing or writing parallel programs that they stop they continue think in terms of the sequential programming style.

This they say here is the thread it does this start on the other thing not realizing or not ignoring that at the same time some other thread is doing things with your variable right. So if you simply see a program you say this variable gets updated here and here if that variable was shared by somebody else and you did not account for that it not going to get updated just here and there okay

And so you have to be extremely mindful all the time of the concurrent session and that is where the majority of bugs come from. As a matter of fact I always say that you have to think of other threads always as adversaries rather than thread co-operating to solve the same problem in the sense that if you are going to assume that thread is going to play nice it is definitely not going to play nice okay

Every race condition that you think is rare is going to come in program okay when you run it or test it or specially if you demonstrate it to somebody else all right those are rules of the programming. So think concurrent at every time you are not only thinking in terms of what I am doing and what I as one of many is doing there are other N -1 things doing whatever they are doing.

And then the other part is the synchronization once I have figured out that these are shared and these depend on these other things then how they should communicate how you should manage their axis how you should stop one from accessing the other or how you should make other processes threads whatever that will do this for you the management of shared resource synchronization or communication of data and so on and so forth.

It exists in the so many of these will exist in the sequential programming but in the concurrent domain very poor and that actually brings us to the past point here which is you employ high level constructs as and when you can do so okay there are some high level constructs which is about automating some of these and open MP basically has lot of high level constructs we say that this stuff should happen in parallel and I do not want to deal with how they talk to each other and high level constructs not only help you make programming simpler.

But also you know that has been debugged you can safely use it and worry about those raise conditions at least in that part of the code okay the problem is that not everything can be or at least as this the state of the art is not everything can be handled can be handled by high level constants okay. And so you would like to let the compiler deal with it I write a sequential program I tell the compiler to run it on this architecture which has so many processors so much memory in the other stuff and it compiles it.

In some cases especially in the research domains some success has been demonstrated but it far from actually been applicable for useful in larger applications in real context. So there will be typically some mixture of high level constructs and user coded careful parallel code that will eventually make the system faster than just using parallel constants everywhere. In fact the cases where you will not be able to use parallel construct at all okay.

So as the slide says there have been significant and mile stone if attempts for compiler to get some parallelism extracted but typically the context of what the application is trying to do what the architecture is become so complex that it is hard for it to be managed in an automatic way okay. And so in variably you are going to have to help the compiler even when you are using it right. So even when I said open MP has these high level constructs by the fact that they are constants tell you that you have to call them right.

It is not inferring from your sequential code how to make it parallel you have to say this region have to he parallelized in this region. You are not doing all the way but you are still providing what runs in parallel what is independent of each other and here are these ten things that can be done at the same time but them this 11 things has to happen after all those 10 things have to be done.

So compiler although would be ideally suited or we would like we would want compiler to be able to take care of the parallel programming problem for us. But it at this as the state of this art is it or not and so here we are and which means that when you start the program you use the program as much high constructs is possible but at the level of the application as the programmer of the application you will be responsible for managing the parallelism what are the things that runs in parallel?

What is the level of parallelism? What is the management of important resources? So there are some resources that can as you see some resources can be managed on automatically that some resources that you are going to have to manage yourself and of course so those last two items basically say that you have got to think in parallel instead of starting to think of here is one program one sequence of instructions.

And you have got to understand how parallel access to data structures and parallel algorithms to do simple things like addition to somewhat less simple things like sorting to an even more complex thing can be done okay so let us stop at this point.